



Bash contra la shell de Windows Vista

SHELLS A PRUEBA

La nueva PowerShell de Microsoft hace uso de librerías de la estructura de .NET, lo que le permite el acceso a un tesoro oculto de funciones y objetos. ¿Se puede medir PowerShell con otras shells tradicionales como Bash? **POR MARKUS NASAREK**

Tanto Bash como la PowerShell de Windows Vista incluyen comandos para moverse por los directorios, gestionar archivos o lanzar otros programas. Una shell debe ser capaz de administrar un sistema, y ambas están equipadas para poder hacerlo desde la línea de comandos.

Mientras que Bash típicamente hace uso de una combinación de herramientas nuevas y clásicas de Unix, PowerShell usa su propio juego de programas para la línea de comandos. Windows se refiere a los comandos de PowerShell como cmdlets. El cmdlet de PowerShell llamado *Get-Process* es un homólogo de *ps*, mientras que *Get-Content* lo es de *less*. PowerShell dista mucho de las antiguas shells de Windows. En este artículo mostraré una comparativa entre Bash y la PowerShell de Windows Vista.

Para el soporte de control de programas, una shell necesita elementos para la evaluación y ejecución de condiciones. *for* o *while* evalúan una variable para proporcionar un número determinado de iteraciones. El Listado 1 compara la salida de un contador con *for* en PowerShell y en Bash.

Bash y PowerShell son similares en cuanto al control del flujo de programas

basados en argumentos con *if* o *switch*. La definición de función, el uso de variables de entorno, el ámbito de variables (restringir la validez de una variable), el uso de expresiones regulares, y la evaluación de los valores de salida de un programa también son similares en ambas.

Permisos Restringidos en la Ejecución

La primera diferencia entre Bash y PowerShell se da a la hora de ejecutar un script. PowerShell no ejecuta scripts por omisión, sólo permite un uso interactivo. De todas formas sí que ejecuta scripts si están firmados digitalmente. La firma digital identifica al autor del script, ya que él es la única persona capaz de crear la firma haciendo

uso de medios criptográficos. Aceptar el certificado de firma digital del autor implica la confianza en el mismo. PowerShell en ningún caso ejecutará scripts “desconocidos”. Para poder ejecutar un script sin firma digital, hay que cambiar la política predeterminada de ejecución a *RemoteSigned* en la línea de comandos, tal que:

```
PS:> Set-ExecutionPolicy RemoteSigned
```

Este comando indica a PowerShell que debe aceptar todos los scripts locales. Si hemos descargado los datos, o si venían adjuntos a un email, la shell insistirá en la firma. Estas nuevas políticas para el desarrollo en materia de seguridad hacen

Listado 1: Bucles

Un bucle for en la PowerShell de Windows Vista:	El mismo bucle con Bash
PS:> for (\$i=1;\$i -le 3;\$i++)	bash[~]\$ for ((i=1;i<=3;i++))
{ Write-Host \$i }	do echo \$i done
1	1
2	2
3	3
PS:>	bash[~]\$

Listado 2: Salida de Get-Member

```
01 PS:> Get-Content Textdatei.txt | Get-Member
02  Typename: System.String
03 Name      MemberType  Definition
04 ----      -
05 Clone     Method           System.Object Clone()
06 CompareTo Method           System.Int32 CompareTo(Object value),
    System.Int32 CompareTo(String strB)
07 Contains  Method           System.Boolean Contains(String value)
08 CopyTo    Method           System.Void CopyTo(Int32 sourceIndex, Char[]
    destination, Int32 destinationIn...
09 ...
10 Length   Property
11 PS:>
```

que virus como el de “Love Letter” [1], basados en scripting, pierden así su efectividad.

A diferencia de PowerShell, Bash no hace uso de la firma digital para evaluar los permisos de ejecución de un script. Son los permisos del sistema de ficheros los que determinan si un script se puede ejecutar o no.

Ambas shells comparten sorprendentes similitudes en la forma en que manejan la configuración del sistema. PowerShell considera todo como un sistema de ficheros moviéndose no sólo por éste y los dispositivos, sino también por el registro, por el lugar donde se almacenan los certificados y también por las variables de entorno, algo completamente nuevo para el mundo de Windows. Podemos copiar, mover y renombrar valores del registro igual que

hacemos con archivos. PowerShell se refiere a estos sistemas de ficheros virtuales como *Providers*, implementando así una filosofía que Linux siempre ofreció a Bash: “Todo es un archivo.”

Redirigir

La herramienta más potente de PowerShell es la tubería. Ésta soporta el paso ordenado de valores, permitiendo el uso de la salida de un comando como entrada para el siguiente. Bash también soporta tuberías, pero no hace solicitudes particulares a archivos de entrada y salida. Confía en que el siguiente comando de la cadena sabe hacer algo con la salida que se le procura.

En PowerShell, todos los cmdlets crean objetos definidos para la salida de datos en lugar de una salida de sólo texto. Aunque parezca que sólo se le están pasando cade-

nas de caracteres en la salida, lo que realmente ocurre es que la salida completa se convierte a un objeto.

Se puede consultar un objeto con el comando *Get-Member*, que muestra sus elementos y funciones. Véase el Listado 2. Por ejemplo, el comando

```
PS:> Get-Content Textdatei.txt |
Sort { $_.Length }
```

nos permite ordenar las líneas en un fichero de texto usando la propiedad *Length* del objeto.

Aunque pasar objetos de datos es un poco más complicado, esta orientación a los objetos ayuda a estandarizar las operaciones y permite el manejo de estructuras de datos complejas. Con esto no compete Bash, el cual hace uso de programas externos que sí que pueden manejar este tipo de estructuras de datos.

Por ejemplo, Bash usa un parseador de XML externo (como Saxon o Xalan-J) para analizar archivos de tipo XML. El Listado 3 es un pequeño script para PowerShell que obtiene un suministro RSS desde Internet y lo guarda en forma de archivo XML. El script define una función *Show-SpiegelRSS* que obtiene los suministros RSS actuales.

Conclusiones

Por un lado, PowerShell recoge el concepto Unix de que son preferibles muchas aplicaciones pequeñas a una grande. Al mismo tiempo adopta, una conducta orientada a objetos, que hace más simple un proyecto de grandes dimensiones, con el coste que supone añadir nuevos pasos a la curva de aprendizaje. El mayor problema con los objetos se produce cuando tenemos que invertir un tiempo significativo en descubrir la función o el objeto que necesitamos. El cmdlet *Get-Member* probablemente será uno de los comandos más usados con PowerShell.

Bash es útil como herramienta directa, para las tareas del día a día. Si sobreviene la necesidad de usos avanzados y estructuras de datos complejas, podemos desviarnos un poco hacia Python orientado a objetos o las capacidades gráficas de Tcl/Tk.

Listado 3: Show-SpiegelRSS.ps1

```
01 # Show-RSS.ps1
02 # Declaración de variables para URL
03 $feed="http://rss.new.yahoo.com/rss/linux"
04 Write-Host -ForegroundColor "green" "RSS-Feed: " $feed
05 # Descarga del suministro de RSS
06 $wco = New-Object System.Net.WebClient
07 $rss = [xml]$wco.DownloadString($feed)
08 # Muestra el título
09 Write-Host -ForegroundColor "red" $rss.rss.channel.title
10 # Muestra un pequeño formulario
11 $rss.rss.item | Select-Object title,description | format-table
12 # Muestra el título y la descripción de las entradas
13 $rss.rss.channel.item | Select-Object title,pubDate,description |
```

RECURSOS

[1] CERT-Avisory CA-2000-04 “Gusano Love Letter”: <http://www.cert.org/advisories/CA-2000-04.html>